МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра информатики и программирования

ИССЛЕДОВАНИЕ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЕРА

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 441 группы
направления 02.03.03 Математическое обеспечение и администрирование
информационных систем
факультета компьютерных наук и информационных технологий
Амангалиева Артема Леонидовича

Научный руководитель:
доцент _____ Е.В. Кудрина

подпись, дата

Зав. кафедрой:
к.ф.-м.н., доцент _____ М.В. Огнева

подпись, дата

Саратов 2022

ВВЕДЕНИЕ

Актуальность темы. Задача коммивояжёра насчитывает более чем двухсотлетнюю историю и является одной из наиболее интенсивно изучаемых проблем оптимизации. Многие знаменитые ученые прошлого, такие как Эйлер, Гамильтон, а также выдающиеся математики современности, предлагали методы ее решения.

Задача коммивояжера имеет практическое применение в различных областях, таких как планирование, логистика, а именно: маршрутизация транспортных потоков; в астрономии, минимизация времени, затрачиваемое на перемещение телескопа, а подвиды задачи коммивояжера помогают в решении задач эффективного сжатия произвольных данных и анализа последовательностей ДНК, эвристического определения схожести строк, построения практических алгоритмов исследования специально определённых бесконечных грамматических структур и построения эволюционных деревьев.

Цель выпускной квалификационной работы: исследование методов решения задачи коммивояжера на примере алгоритмов полного перебора, ближайшего соседа, ветвей и границ, реализованных на языке Python.

Поставленная цель определила следующие задачи:

1. Рассмотреть историю задачи коммивояжера.
2. Описать классификацию методов решения задачи коммивояжера.
3. Изучить инструментальные средства и технологий, используемые для реализации методов решения задачи коммивояжера.
4. Реализовать методы полного перебора, ближайшего соседа, ветвей и границ используя язык Python.
5. Провести локальную оптимизацию вышеописанных алгоритмов, используя библиотеки языка Python.
6. Представить сравнительный анализ данных методов.

Методологические основы решения задачи коммивояжера представлены в работах Мудрова В. И.[1], Гудмана С. и Хидетниеми С.[2], Дулькейта В. И. и

Файзуллина Р.Т.[3], Ермоленко С.В и Кобака В.Г.[3], Ерзин А.И. и Кочетов Ю.А. [5], Гимади Э.Х., Глебова Н.И. и Сердюков А.И.[6].

Практическая значимость бакалаврской работы заключается в реализации и исследовании методов полного перебора, ближайшего соседа, ветвей и границ используя язык Python, проведения локальной оптимизации, а также проведения сравнительного анализа методов решения задачи коммивояжера.

Структура и объём работы. Бакалаврская работа состоит из введения, двух разделов, заключения, списка использованных источников и двух приложений. Общий объем работы – 52 страниц, из них 45 страниц – основное содержание, включая 24 рисунка и 9 таблиц, цифровой носитель в качестве приложения, список использованных источников информации – 21 наименование.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Теоретическая часть» посвящен классификации методов решения задачи коммивояжера, а также постановке задачи и ее истории, приведен обзор инструментальных средств, необходимых для реализации цели и задач работы.

1.1 Постановка задачи коммивояжера

Задача коммивояжера была математически описана в 19 веке ирландским математиком Уильямом Роуэном Гамильтоном и британским математиком Томасом Киркманом. Икосианская игра Гамильтона была развлекательной головоломкой, основанной на поиске гамильтонового цикла. Гамильтон придумал игру, в которой игрокам необходимо было составить тур через 20 точек, используя только нарисованные связи между точками (фигура соответствовала додекаэдру).

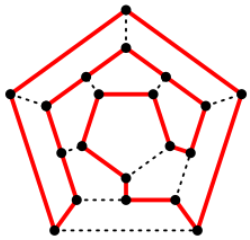


Рисунок 1 – Решение игры, показанное красной линией, которая посещает каждую вершину додекаэдра и образует гамильтонов цикл.

Общая форма задача, была впервые рассмотрена математиками в 1930–х годах в Вене и в Гарварде, в частности Карлом Менгером, который определяет проблему, рассматривает очевидный алгоритм грубой силы и наблюдает неоптимальность эвристики ближайшего соседа.

Задачу коммивояжера определим несколькими способами:

1. Классическая постановка.

Имеется n городов, которые должен обойти коммивояжер с максимальными затратами. При этом на его маршрут накладывается два ограничения:

- маршрут должен быть замкнутым, то есть коммивояжер должен вернуться в тот город, из которого он начал движение;
- в каждом из городов коммивояжер должен побывать точно один раз, то есть надо обязательно обойти все города, при этом не побывав ни в одном городе дважды.

2. В терминах графов.

Пусть дан обыкновенный связный взвешенный граф (веса ребер заданы матрицей $\{c_{ij}\}$). Требуется найти гамильтонов цикл наименьшего веса, где вес цикла определяется как сумма весов входящих в него ребер.

1.2 Методы решение задачи коммивояжера

Существует множество различных методов решения задачи коммивояжера.

1. Точные методы: Метод полного перебора

2. Эвристические методы: Метод ближайшего соседа, Алгоритм Метрополиса (имитация отжига), Метод локальных улучшений, Метод Монте – Карло (метод случайного перебора), Использование генетических алгоритмов, Метод муравьиных колоний, Метод ветвей и границ.

Все эффективные (сокращающие полный перебор) методы решения TSP – методы эвристические. В большинстве эвристических методов происходит поиск не самого эффективного маршрута, а находится приближённое решение.

Таблица 1 – Асимптотическая сложность описанных алгоритмов

Метод решения задачи коммивояжера	Асимптотическая сложность
Полный перебор	$O(n!)$

Ветвей и границ	$O(n\ log\ n)$
Ближайшего соседа	$O(n)$
Генетический	$O(n^2)$
Муравьиной колонии	$O(n^2)$

Для реализации методов решения задачи коммивояжера были выбраны три метода: полного перебора, ближайшего соседа, ветвей и границ. Методы были выбраны исходя из точности результата и фундаментальности метода, быстроты и простоты реализации, оптимальной точности и относительной быстроты, для методов полного перебора, ближайшего соседа, ветвей и границ соответственно.

1.3 Обзор инструментальных средств и технологий, используемых для реализации методов решения задачи коммивояжера.

Python – интерпретируемый, объектно–ориентированный высокоуровневый язык программирования с динамической семантикой.

Matplotlib – это кроссплатформенная библиотека визуализации данных, построенная на массивах NumPy и предназначенная для работы с более широким стеком SciPy.

NumPy, что расшифровывается как Numerical Python, представляет собой библиотеку, состоящую из многомерных объектов массива и набора процедур для обработки этих массивов. С помощью NumPy можно выполнять математические и логические операции с массивами.

Tkinter. Python имеет много графических фреймворков, но Tkinter является единственным фреймворком, встроенным в стандартную библиотеку Python. У Ткинтера есть несколько сильных сторон. Он кроссплатформенный, поэтому один и тот же код работает в Windows, macOS и Linux.

Модуль time из стандартной библиотеки языка программирования Python содержит массу полезных методов для работы со временем.

Модуль math Python является важной функцией, предназначенной для работы с математическими операциями.

Модуль tracemalloc является инструментом отладки для отслеживания блоков памяти, выделенных Python.

PyCharm – это интегрированная среда разработки (IDE), используемая в компьютерном программировании, специально для языка программирования Python.

Второй раздел «Практическая часть» посвящен реализации и исследованию методов решения задачи коммивояжера.

2.1 Реализация метода решения задачи коммивояжера методом полного перебора и его оптимизация

Для реализации точного метода решения задачи коммивояжера методом полного перебора, а в последующем и других методов решения задач коммивояжера, был разработан класс Matrix.

```
class Matrix:
    def __init__(self):
        self.matrix = 0
        self.matrix_size = 0
        self.x = 0
        self.Y = 0

    def generate_matrix(self, n):...
```

Рисунок 2 – Структура класса Matrix

Для реализации переборного метода решения задачи коммивояжера методом полного перебора (brute force), были разработаны три метода.

```
def brute_force(cities):...

def dist(x1, y2):...

def calc_length(cities, path):...
```

Рисунок 3 – Методы необходимы для полного перебора

- Метод brute_force – реализует алгоритм полного перебора.
- Метод dist вычисляет Евклидово расстояние между двумя вершинами.
- Метод calc_length вычисляет полное расстояние для каждого варианта маршрута.

Произведена оптимизация метода полного перебора. Будем использовать встроенную структуру данных list взамен структуры данных ndarray из библиотеки Numpy (изменения в симметричной матрице расстояний). Также будем использовать метод math.pow( ), для возведения в степень в методе dist.

В исследовательских целях был проведен замер времени выполнения алгоритма полного перебора. Замеры проводились только на участке кода самого алгоритма, чтобы объективно оценить скорость его работы.

Таблица 2 – Оценка времени работы метода полного перебора

Входные данные (количество вершин)	Метод полного перебора, сек.
5	0.00099945
6	0.00999022
7	0.08494663
8	0.86346531
9	8.8965044
10	86.15776801

Как видно из таблицы, уже при 10 вершинах, время работы метода существенно увеличилось, по сравнению тем, когда вершин было 9. Это связано с факториальным временем работы алгоритма полного перебора.

Произведем оптимизацию метода полного перебора. Будем использовать встроенную структуру данных list взамен структуры данных ndarray из библиотеки Numpy (изменения в симметричной матрице расстояний). Также будем использовать метод math.pow( ), для возведения в степень в методе dist.

Таблица 3 – Сравнение работы алгоритмов

Входные данные (количество вершин)	Метод полного перебора, сек.	Оптимизированный метод полного перебора, сек.
5	0.00099945	0.00099993
6	0.00999022	0.00599599
7	0.08494663	0.03997278
8	0.86346531	0.36677384
9	8.8965044	3.82878637
10	86.15776801	43.09669924

Из таблицы №3 можно сделать вывод, что реализация с использованием встроенной структуры данных list работает примерно в 2 раз быстрее. Но, в общем и целом, данная оптимизация не эффективна, так как асимптотическая сложность для графа с большим числом городом скажется на времени работы в худшую сторону, ввиду перебора всевозможных вариантов.

2.2 Реализация приближенного метода решения задачи коммивояжера методом ближайшего соседа и его оптимизация

Для решения задачи коммивояжера был разработан класс NearestNeighbor. В данном классе были созданы два метода nearest_neighbor() и nearest_neighbor_optimized(), которые решают задачу коммивояжера методом ближайшего соседа и модифицированным методом ближайшего соседа.

Рисунок 4 – Структура класса NearestNeighbor

Произведена оптимизацию двух вышеупомянутых методов. Будем использовать структуру данных ndarray из библиотеки Numpy взамен встроенной структуры данных list, при генерации матрицы. Структура ndarray предназначена для сложных математических вычислений с большим объемом данных. Также будем использовать метод и math.pow( ), для возведения в степень.

Для сравнения скорости выполнения двух алгоритмов был использован таймер из модуля timeit для замера времени выполнения участков кода. Замеры проводились только на участках кода самих алгоритмов, чтобы объективно оценить скорость их работы.

Таблица 4 – Сравнение работы алгоритмов ближайшего соседа

Входные данные (количество вершин)	Метод ближайшего соседа		Модифицированный метод ближайшего соседа	
	Время, сек.	Сумма маршрута	Время, сек.	Сумма маршрута
10	0,000567	304,98	0,003676	304,98
25	0,001105	380,78	0,029210	376,66
50	0,005363	666,27	0,172270	602,76
100	0,011957	994,73	1,172542	888,55
250	0,070147	1515,89	15,696372	1407,67
500	0,272049	2056,09	133,799222	1947,73
1000	1,177980	3023,19	2809,091842	2809,65

Как видно из таблицы, модифицированный метод находит маршрут с меньшей суммой, но тратит существенно больше времени, чем классический.

Произведем оптимизацию двух вышеупомянутых методов. Будем использовать структуру данных ndarray из библиотеки Numpy взамен встроенной структуры данных list, при генерации матрицы. Структура ndarray предназначена для сложных математических вычислений с большим объемом данных. Также будем использовать метод и math.pow( ), для возведения в степень.

Таблица 5 – Сравнение работы модифицированных алгоритмов

Входные данные (количество вершин)	Метод ближайшего соседа			Модифицированный метод ближайшего соседа		
	Неоптимизированный вариант, сек.	Оптимизированный вариант, сек.	Сумма маршрута	Неоптимизированный вариант, сек.	Оптимизированный вариант, сек.	Сумма маршрута
10	0,000567	0,000224	304,98	0,003676	0,001663	304,98
25	0,001105	0,000573	380,78	0,029210	0,012320	376,66
50	0,005363	0,001523	666,27	0,172270	0,065530	602,76
100	0,011957	0,003242	994,73	1,172542	0,369042	888,55
250	0,070147	0,034685	1515,89	15,696372	4,368548	1407,67
500	0,272049	0,073840	2056,09	133,799222	33,739510	1947,73
1000	1,177980	0,241985	3023,19	2809,091842	323,190016	2809,65
5000	29,862869	6,638811	6404.51	–	–	–

2.3 Реализация приближенного метода решения задачи коммивояжера методом ветвей и границ и его оптимизация

В реализации приближенного метода решения задачи коммивояжера методом ветвей будет использовать класс Matrix из предыдущего раздела, для хранения и манипуляций с расстояниями между точками графа.

Реализуем класс Node для хранения узлов графа.

```
class Node:
    def __init__(self):
        self.value = 0
        self.vertex = 0
        self.x = None
        self.y = None
```

Рисунок 5 – Структура класса Node

Опишем поля данного класса:

- Поле value – будет хранить значение пути.
- Поле vertex – будет ссылку на вершину.

Методы класса Node:

- Метод set_x_and_y задает значение координатам x и y.
- Метод set_vertex задает значение вершины.
- Метод set_nums_col_and_row для правильного отображения путей нам необходима матрица со всеми номерами строк и столбцов, так как при выполнении метода ветвей и границ на каждой итерации вычеркивается найденная строка и столбец в матрице расстояний между точками.

Класс BranchAndBound. Данный класс содержит реализацию метода ветвей и границ, а также вспомогательных методов.

Рисунок 6 – Структура класса BranchAndBound

Произведена оптимизация метода ветвей и границ. Будем использовать встроенную структуру данных list взамен структуры данных ndarray из библиотеки Numpy, при генерации матрицы расстояний. Заменим реализации методов find_min_element_in_row, find_min_element_in_col, cut_matrix с использованием Numpy, на реализацию проделывающие такие же манипуляции, но для списков.

Был проведен замер времени выполнения алгоритма метода ветвей и границ. Замеры проводились только на участке кода самого алгоритма, чтобы объективно оценить скорость его работы.

Таблица 6 – Оценка времени работы метода ветвей и границ

Входные данные (количество вершин)	Метод ветвей и границ	
	Время, сек.	Сумма маршрута
10	0,00868	256.43
25	0,13645	363.39
50	0,60458	564.97
100	5,17112	801.52
150	19,52887	1243.57
250	113,71259	1452.82

Произведем оптимизацию метода ветвей и границ. Будем использовать встроенную структуру данных list взамен структуры данных ndarray из библиотеки Numpy, при генерации матрицы расстояний. Заменим реализации методов find_min_element_in_row, find_min_element_in_col, cut_matrix с использованием Numpy, на реализацию проделывающие такие же манипуляции, но для списков.

Таблица 7 – Сравнение работы алгоритмов

Входные данные (количество вершин)	Метод ветвей и границ, сек.	Оптимизированный метод ветвей и границ, сек.
10	0,00868	0,00078
25	0,13645	0,00556
50	0,60458	0,03425
100	5,17112	0,31543
150	19,52887	2,21324
250	113,71259	4,54412
500	–	46,6494

Из таблицы 7 можно сделать вывод, что реализация с использованием встроенной структуры данных list работает примерно в 20 раз быстрее.

2.4 Сравнительный анализ оптимизированных методов решения задачи коммивояжера

Сравним работу оптимизированных методов решения задачи коммивояжера на одинаковом наборе данных.

Таблица 8 – Сравнение работы алгоритмов по времени

Входные данные (количество вершин)	Оптимизированный метод полного перебора, сек.	Оптимизированный метод ближайшего соседа, сек.	Оптимизированный метод ветвей и границ, сек.
5	0.00099945	0,000132	0,0002
10	43.09669924	0,000224	0,00078
25	–	0,000573	0,00556
50	–	0,001523	0,03425
100	–	0,003242	0,31543
250	–	0,034685	4,54412
500	–	0,073840	46,6494
1000	–	0,241985	–
5000	–	6,638811	–

Проанализируем алгоритмы на точность маршрута, построенного ими. На маленьком количестве вершин – равным 5, метод ветвей и границ и метод ближайшего соседа показывают точные результаты, но в общем случае ввиду того, что методы приближенные, результаты могут быть не точны. Из–за того, что метод полного перебора работает за факториал, проанализировать маршрут данным алгоритмом, ввиду времени и мощностей компьютера не представляется возможным.

Таблица 9 – Сравнение работы алгоритмов по точности маршрута

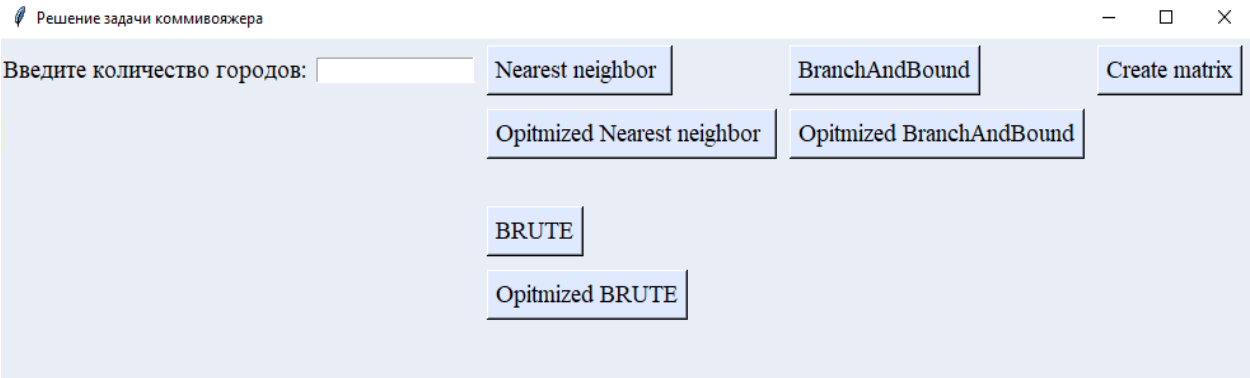
Входные данные (количество вершин)	Оптимизированный метод полного перебора, сек.	Оптимизированный метод ближайшего соседа, сумма маршрута	Оптимизированный метод ветвей и границ, сумма маршрута
5	239.6	239.6	239.6

10	286.62	291.44	305.12
25	–	522.41	408.67
50	–	720.36	652.49
100	–	883.61	813.97
250	–	1522.2	1301.03
500		2056,09	1995.67

Можно сделать вывод, что метод полного перебора необходим при критичности знания точного результата маршрута, и при большой вычислительной возможности компьютера или сервера. Приближенные методы подойдут в приложения, где необходимы быстрые расчеты результата маршрута и не так важна точность результата.

2.5 Интерфейс приложения

Используя библиотеку tkinter, был разработан и реализован графический интерфейс для пользователей для удобной работы с приложением.



ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы бакалавра были решены все поставленные задачи, что позволило исследовать такие методы решения задачи коммивояжера как алгоритм полного перебора, алгоритм ближайшего соседа, алгоритм ветвей и границ. Следует отметить, что в процессе реализации данных алгоритмов на языке Python была их выполнена программная оптимизация.

Проведенные исследования показали, что метод полного перебора необходим при критичности знания точного результата маршрута, и при большой вычислительной возможности компьютера или сервера, а также небольшом количестве “городов”. Приближенные методы подойдут в прило-

жения, где необходимы быстрые расчеты результата маршрута, где полученный результат в некоторой окрестности будет удовлетворять требованиям задачи.

Данное исследование можно продолжить с использованием более современных алгоритмов задачи коммивояжера и облачных вычислений.

Основные источники информации:

1. Задача о коммивояжере. Мудров В. И. Издательство “Знание”. Москва 1969 год.
2. Введение в разработку и анализ алгоритмов: учебное пособие. С. Гудман, С. Хидетниеми. Перевод с английского Ю.Б. Котова, Л.В. Сухаревой, Л.В. Ухова под редакцией В.В. Мартынюка. – М: Мир, 1981.
3. Приближённое решение задачи коммивояжера методов рекурсивного построения вспомогательной кривой. В.И. Дулькейт, Р. Т. Файзуллин. ПДМ. 2009.
4. Исследования решения задачи коммивояжера с помощью островной модели. С.В. Ермоленко, В. Г. Кобак, Ростов-на-Дону. 2016.
5. Задачи маршрутизации: учеб. Пособие. А.И. Ерзин, Ю.А. Кочетов. – Новосиб. гос ун-т. – Новосибирск: РИЦ НГУ, 2014.
6. Алгоритм для приближенного решения задачи коммивояжера и его вероятностный анализ. Э.Х. Гимади, Н.И. Глебов, А.И. Сердюков. – Сиб. журн. исслед. операций, 1994